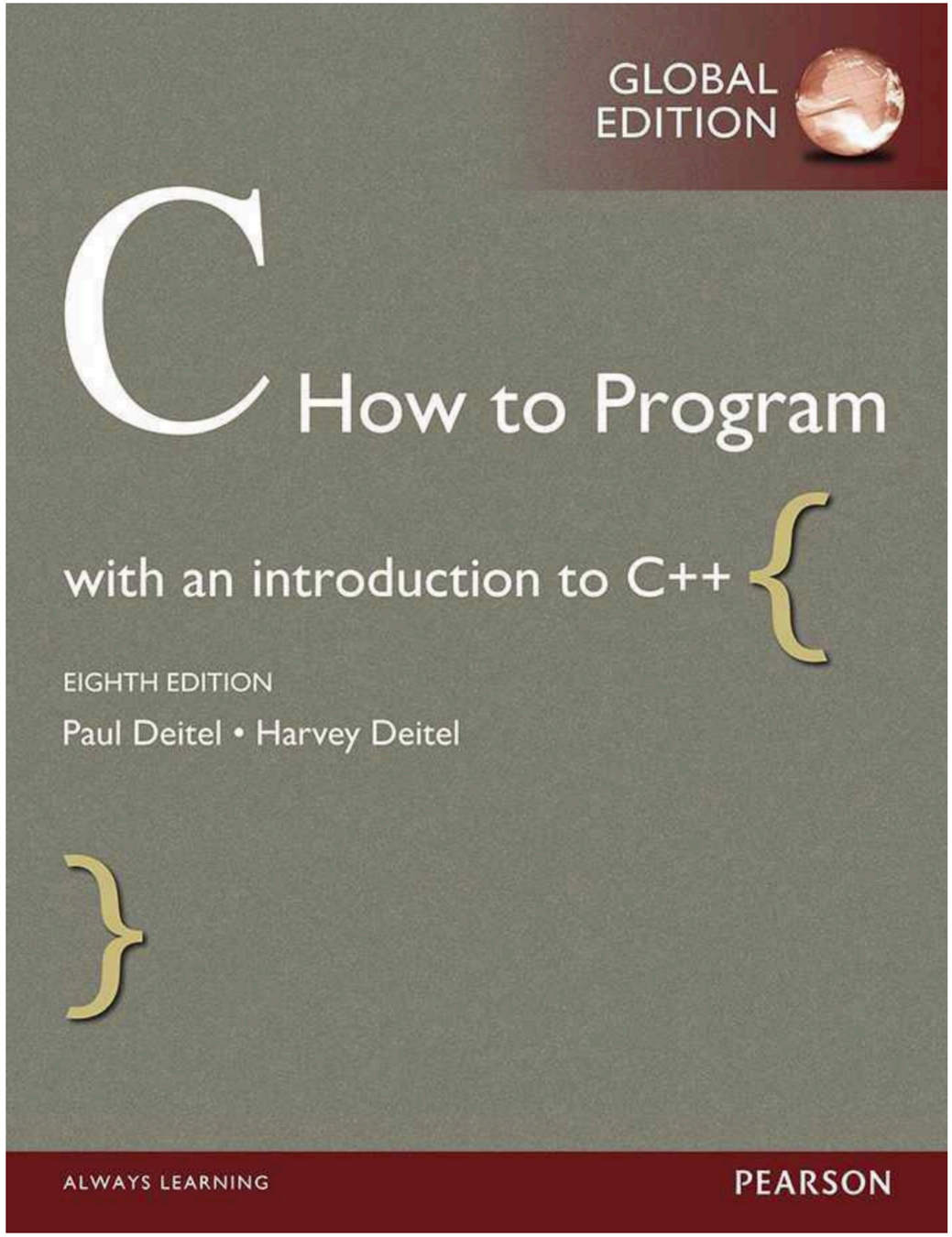
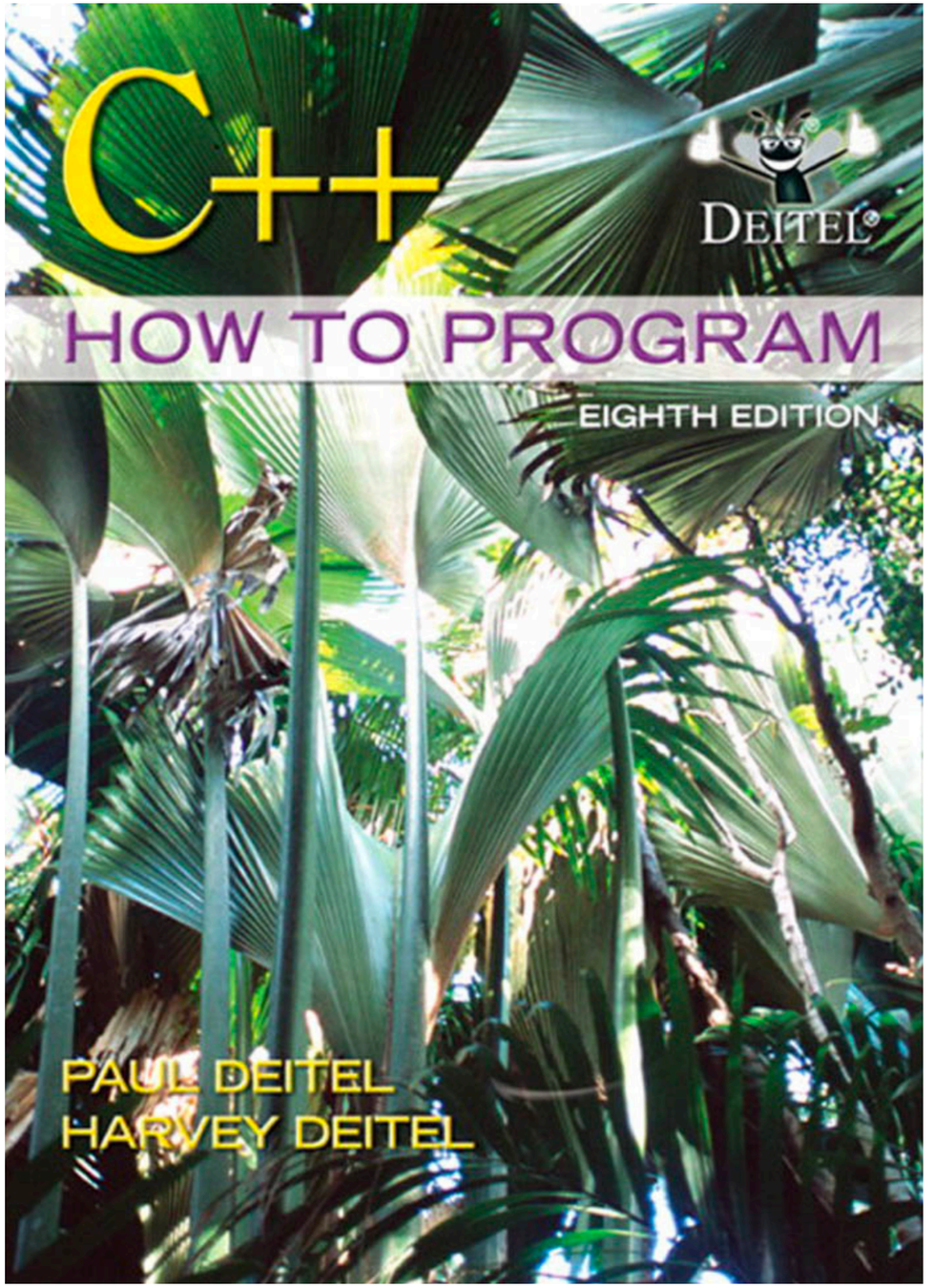
C++

Goshgar Can Ismayilov


GLOBAL EDITION
C
How to Program
with an introduction to C++
EIGHTH EDITION
Paul Deitel • Harvey Deitel
ALWAYS LEARNING
PEARSON


C++
DEITEL®
HOW TO PROGRAM
EIGHTH EDITION
PAUL DEITEL
HARVEY DEITEL

Simple C++ Program

```
1  // Fig. 15.1: fig15_01.cpp
2  // Addition program that displays the sum of two numbers.
3  #include <iostream> // allows program to perform input and output
4
5  int main()
6  {
7     int number1; // first integer to add
8
9     std::cout << "Enter first integer: "; // prompt user for data
10    std::cin >> number1; // read first integer from user into number1
11
12    int number2; // second integer to add
13    int sum; // sum of number1 and number2
14
15    std::cout << "Enter second integer: "; // prompt user for data
16    std::cin >> number2; // read second integer from user into number2
17    sum = number1 + number2; // add the numbers; store result in sum
18    std::cout << "Sum is " << sum << std::endl; // display sum; end line
19 } // end function main
```

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Inline Function

```
1  // Fig. 15.3: fig15_03.cpp
2  // inline function that calculates the volume of a cube.
3  #include <iostream>
4  using std::cout;
5  using std::cin;
6  using std::endl;
7
8  // Definition of inline function cube. Definition of function appears
9  // before function is called, so a function prototype is not required.
10 // First line of function definition acts as the prototype.
11 inline double cube( const double side )
12 {
13    return side * side * side; // calculate the cube of side
14 } // end function cube
15
16 int main()
17 {
18    double sideValue; // stores value entered by user
19
20    for ( int i = 1; i <= 3; i++ )
21    {
22       cout << "\nEnter the side length of your cube: ";
23       cin >> sideValue; // read value from user
24
25       // calculate cube of sideValue and display result
26       cout << "Volume of cube with side "
27          << sideValue << " is " << cube( sideValue ) << endl;
28    }
29 } // end main
```

```
Enter the side length of your cube: 1.0
Volume of cube with side 1 is 1

Enter the side length of your cube: 2.3
Volume of cube with side 2.3 is 12.167

Enter the side length of your cube: 5.4
Volume of cube with side 5.4 is 157.464
```

Pass by Value and Reference

```
x = 2 before squareByValue
Value returned by squareByValue: 4
x = 2 after squareByValue

z = 4 before squareByReference
z = 16 after squareByReference
```

```
 9  int main()
10  {
11     int x = 2; // value to square using squareByValue
12     int z = 4; // value to square using squareByReference
13
14     // demonstrate squareByValue
15     cout << "x = " << x << " before squareByValue\n";
16     cout << "Value returned by squareByValue: "
17        << squareByValue( x ) << endl;
18     cout << "x = " << x << " after squareByValue\n" << endl;
19
20     // demonstrate squareByReference
21     cout << "z = " << z << " before squareByReference" << endl;
22     squareByReference( z );
23     cout << "z = " << z << " after squareByReference" << endl;
24  } // end main
25
26  // squareByValue multiplies number by itself, stores the
27  // result in number and returns the new value of number
28  int squareByValue( int number )
29  {
30     return number *= number; // caller's argument not modified
31  } // end function squareByValue
32
33  // squareByReference multiplies numberRef by itself and stores the result
34  // in the variable to which numberRef refers in the caller
35  void squareByReference( int &numberRef )
36  {
37     numberRef *= numberRef; // caller's argument modified
38  } // end function squareByReference
```

Default Function Arguments

```
6  // function prototype that specifies default arguments
7  int boxVolume( int length = 1, int width = 1, int height = 1 );
8
9  int main()
10 {
11    // no arguments--use default values for all dimensions
12    cout << "The default box volume is: " << boxVolume();
13
14    // specify length; default width and height
15    cout << "\n\nThe volume of a box with length 10,\n"
16       << "width 1 and height 1 is: " << boxVolume( 10 );
17
18    // specify length and width; default height
19    cout << "\n\nThe volume of a box with length 10,\n"
20       << "width 5 and height 1 is: " << boxVolume( 10, 5 );
21
22    // specify all arguments
23    cout << "\n\nThe volume of a box with length 10,\n"
24       << "width 5 and height 2 is: " << boxVolume( 10, 5, 2 )
25       << endl;
26 } // end main
27
28 // function boxVolume calculates the volume of a box
29 int boxVolume( int length, int width, int height )
30 {
31    return length * width * height;
32 } // end function boxVolume
```

```
The default box volume is: 1

The volume of a box with length 10,
width 1 and height 1 is: 10

The volume of a box with length 10,
width 5 and height 1 is: 50

The volume of a box with length 10,
width 5 and height 2 is: 100
```

Unary Scope Resolution

```
Local double value of number = 10.5
Global int value of number = 7
```

```
1  // Fig. 15.9: fig15_09.cpp
2  // Using the unary scope resolution operator.
3  #include <iostream>
4  using namespace std;
5
6  int number = 7; // global variable named number
7
8  int main()
9  {
10    double number = 10.5; // local variable named number
11
12    // display values of local and global variables
13    cout << "Local double value of number = " << number
14       << "\nGlobal int value of number = " << ::number << endl;
15 }
```

Function Overloading

```
square of integer 7 is 49
square of double 7.5 is 56.25
```

```
6  // function square for int values
7  int square( int x )
8  {
9     cout << "square of integer " << x << " is ";
10    return x * x;
11 } // end function square with int argument
12
13 // function square for double values
14 double square( double y )
15 {
16    cout << "square of double " << y << " is ";
17    return y * y;
18 } // end function square with double argument
19
20 int main()
21 {
22    cout << square( 7 ); // calls int version
23    cout << endl;
24    cout << square( 7.5 ); // calls double version
25    cout << endl;
26 } // end main
```

C++ Vectors

```
Size of vector integers1 is 7
vector after initialization:
           0           0           0           0
           0           0           0

Size of vector integers2 is 10
vector after initialization:
           0           0           0           0
           0           0           0           0
           0           0
```

```
1   // Fig. 15.14: fig15_14.cpp
2   // Demonstrating C++ Standard Library class template vector.
3   #include <iostream>
4   #include <iomanip>
5   #include <vector>
6   using namespace std;
7
8   void outputVector( const vector< int > & ); // display the vector
9   void inputVector( vector< int > & ); // input values into the vector
10
11  int main()
12  {
13     vector< int > integers1( 7 ); // 7-element vector< int >
14     vector< int > integers2( 10 ); // 10-element vector< int >
15
16     // print integers1 size and contents
17     cout << "Size of vector integers1 is " << integers1.size()
18        << "\nvector after initialization:" << endl;
19     outputVector( integers1 );
20
21     // print integers2 size and contents
22     cout << "\nSize of vector integers2 is " << integers2.size()
23        << "\nvector after initialization:" << endl;
24     outputVector( integers2 );
```

```
87  // output vector contents
88  void outputVector( const vector< int > &array )
89  {
90     size_t i; // declare control variable
91
92     for ( i = 0; i < array.size(); ++i )
93     {
94        cout << setw( 12 ) << array[ i ];
95
96        if ( ( i + 1 ) % 4 == 0 ) // 4 numbers per row of output
97           cout << endl;
98     }
99
100    if ( i % 4 != 0 )
101       cout << endl;
102 }
103
104 // input vector contents
105 void inputVector( vector< int > &array )
106 {
107    for ( size_t i = 0; i < array.size(); ++i )
108       cin >> array[ i ];
109 }
```

C++ Vectors

```
26    // input and print integers1 and integers2
27    cout << "\nEnter 17 integers:" << endl;
28    inputVector( integers1 );
29    inputVector( integers2 );
30
31    cout << "\nAfter input, the vectors contain:\n"
32       << "integers1:" << endl;
33    outputVector( integers1 );
34    cout << "integers2:" << endl;
35    outputVector( integers2 );
36
37    // use inequality (!=) operator with vector objects
38    cout << "\nEvaluating: integers1 != integers2" << endl;
39
40    if ( integers1 != integers2 )
41       cout << "integers1 and integers2 are not equal" << endl;
42
43    // create vector integers3 using integers1 as an
44    // initializer; print size and contents
45    vector< int > integers3( integers1 ); // copy constructor
46
47    cout << "\nSize of vector integers3 is " << integers3.size()
48       << "\nvector after initialization:" << endl;
49    outputVector( integers3 );
50
51    // use overloaded assignment (=) operator
52    cout << "\nAssigning integers2 to integers1:" << endl;
53    integers1 = integers2; // assign integers2 to integers1
54
55    cout << "integers1:" << endl;
56    outputVector( integers1 );
57    cout << "integers2:" << endl;
58    outputVector( integers2 );
59
60    // use equality (==) operator with vector objects
61    cout << "\nEvaluating: integers1 == integers2" << endl;
62
63    if ( integers1 == integers2 )
64       cout << "integers1 and integers2 are equal" << endl;
65
66    // use square brackets to create rvalue
67    cout << "\nintegers1[5] is " << integers1[ 5 ];
```

```
Enter 17 integers:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

After input, the vectors contain:
integers1:
           1           2           3           4
           5           6           7
integers2:
           8           9          10          11
          12          13          14          15
          16          17

Evaluating: integers1 != integers2
integers1 and integers2 are not equal

Size of vector integers3 is 7
vector after initialization:
           1           2           3           4
           5           6           7

Assigning integers2 to integers1:
integers1:
           8           9          10          11
          12          13          14          15
          16          17
integers2:
           8           9          10          11
          12          13          14          15
          16          17

Evaluating: integers1 == integers2
integers1 and integers2 are equal

integers1[5] is 13
```

C++ Vectors

```
integers1[5] is 13

Assigning 1000 to integers1[5]
integers1:
           8           9          10          11
          12        1000          14          15
          16          17

Attempt to display integers1.at( 15 )
An exception occurred: invalid vector<T> subscript
```

```
69      // use square brackets to create lvalue
70      cout << "\n\nAssigning 1000 to integers1[5]" << endl;
71      integers1[ 5 ] = 1000;
72      cout << "integers1:" << endl;
73      outputVector( integers1 );
74
75      // attempt to use out-of-range index
76      try
77      {
78         cout << "\nAttempt to display integers1.at( 15 )" << endl;
79         cout << integers1.at( 15 ) << endl; // ERROR: out of range
80      }
81      catch ( out_of_range &ex )
82      {
83         cout << "An exception occurred: " << ex.what() << endl;
84      }
85  }
```

Simple C++ Class

```
Please enter the course name:
CS101 Introduction to C++ Programming

Welcome to the grade book for
CS101 Introduction to C++ Programming!
```

```cpp
// GradeBook class definition
class GradeBook
{
public:
   // function that displays a welcome message to the GradeBook user
   void displayMessage( string courseName )
   {
      cout << "Welcome to the grade book for\n" << courseName << "!"
         << endl;
   } // end function displayMessage
}; // end class GradeBook

// function main begins program execution
int main()
{
   string nameOfCourse; // string of characters to store the course name
   GradeBook myGradeBook; // create a GradeBook object named myGradeBook

   // prompt for and input course name
   cout << "Please enter the course name:" << endl;
   getline( cin, nameOfCourse ); // read a course name with blanks
   cout << endl; // output a blank line

   // call myGradeBook's displayMessage function
   // and pass nameOfCourse as an argument
   myGradeBook.displayMessage( nameOfCourse );
} // end main
```

```json
{
    "camelCase": "camelCase",
    "PascalCase": "PascalCase",
    "snake_case": "snake_case",
    "kebab-case": "kebab-case"
}
```

Class - Set/Get Functions

```
Initial course name is:

Please enter the course name:
CS101 Introduction to C++ Programming

Welcome to the grade book for
CS101 Introduction to C++ Programming!
```

```
 9  // GradeBook class definition
10  class GradeBook
11  {
12  public:
13     // function that sets the course name
14     void setCourseName( string name )
15     {
16        courseName = name; // store the course name in the object
17     } // end function setCourseName
18
19     // function that gets the course name
20     string getCourseName()
21     {
22        return courseName; // return the object's courseName
23     } // end function getCourseName
24
25     // function that displays a welcome message
26     void displayMessage()
27     {
28        // this statement calls getCourseName to get the
29        // name of the course this GradeBook represents
30        cout << "Welcome to the grade book for\n" << getCourseName() << "!"
31           << endl;
32     } // end function displayMessage
33  private:
34     string courseName; // course name for this GradeBook
35  }; // end class GradeBook
```

```
37  // function main begins program execution
38  int main()
39  {
40     string nameOfCourse; // string of characters to store the course name
41     GradeBook myGradeBook; // create a GradeBook object named myGradeBook
42
43     // display initial value of courseName
44     cout << "Initial course name is: " << myGradeBook.getCourseName()
45        << endl;
46
47     // prompt for, input and set course name
48     cout << "\nPlease enter the course name:" << endl;
49     getline( cin, nameOfCourse ); // read a course name with blanks
50     myGradeBook.setCourseName( nameOfCourse ); // set the course name
51
52     cout << endl; // outputs a blank line
53     myGradeBook.displayMessage(); // display message with new course name
54  } // end main
```

Class - Constructor

```
gradeBook1 created for course: CS101 Introduction to C++ Programming
gradeBook2 created for course: CS102 Data Structures in C++
```

```
 9  // GradeBook class definition
10  class GradeBook
11  {
12  public:
13     // constructor initializes courseName with string supplied as argument
14     GradeBook( string name )
15     {
16        setCourseName( name ); // call set function to initialize courseName
17     } // end GradeBook constructor
18
```

```
42  // function main begins program execution
43  int main()
44  {
45     // create two GradeBook objects
46     GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
47     GradeBook gradeBook2( "CS102 Data Structures in C++" );
48
49     // display initial value of courseName for each GradeBook
50     cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
51        << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
52        << endl;
53  } // end main
```

Header File

```
gradeBook1 created for course: CS101 Introduction to C++ Programming
gradeBook2 created for course: CS102 Data Structures in C++
```

```
1  // Fig. 16.10: fig16_10.cpp
2  // Including class GradeBook from file GradeBook.h for use in main.
3  #include <iostream>
4  #include "GradeBook.h" // include definition of class GradeBook
5  using namespace std;
6
7  // function main begins program execution
8  int main()
9  {
10    // create two GradeBook objects
11    GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
12    GradeBook gradeBook2( "CS102 Data Structures in C++" );
13
14    // display initial value of courseName for each GradeBook
15    cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
16       << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
17       << endl;
18 } // end main
```

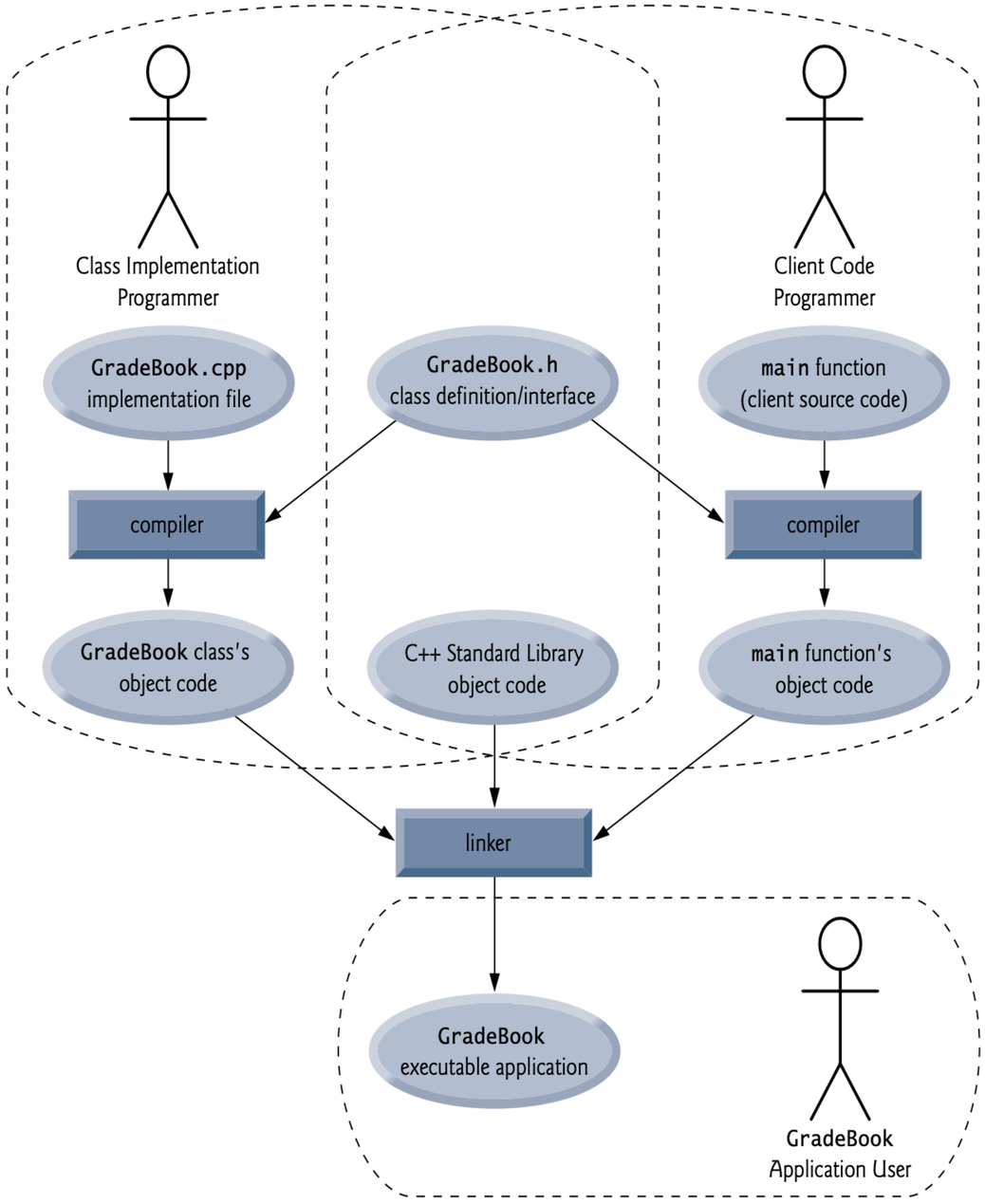
Interface

```
1  // Fig. 16.11: GradeBook.h
2  // GradeBook class definition. This file presents GradeBook's public
3  // interface without revealing the implementations of GradeBook's member
4  // functions, which are defined in GradeBook.cpp.
5  #include <string> // class GradeBook uses C++ standard string class
6  using namespace std;
7
8  // GradeBook class definition
9  class GradeBook
10 {
11 public:
12    GradeBook( string ); // constructor that initializes courseName
13    void setCourseName( string ); // function that sets the course name
14    string getCourseName(); // function that gets the course name
15    void displayMessage(); // function that displays a welcome message
16 private:
17    string courseName; // course name for this GradeBook
18 }; // end class GradeBook
```

```
1  // Fig. 16.10: fig16_10.cpp
2  // Including class GradeBook from file GradeBook.h for use in main.
3  #include <iostream>
4  #include "GradeBook.h" // include definition of class GradeBook
5  using namespace std;
6
7  // function main begins program execution
8  int main()
9  {
10    // create two GradeBook objects
11    GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
12    GradeBook gradeBook2( "CS102 Data Structures in C++" );
13
14    // display initial value of courseName for each GradeBook
15    cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
16       << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
17       << endl;
18 } // end main
```

```
1  // Fig. 16.12: GradeBook.cpp
2  // GradeBook member-function definitions. This file contains
3  // implementations of the member functions prototyped in GradeBook.h.
4  #include <iostream>
5  #include "GradeBook.h" // include definition of class GradeBook
6  using namespace std;
7
8  // constructor initializes courseName with string supplied as argument
9  GradeBook::GradeBook( string name )
10 {
11    setCourseName( name ); // call set function to initialize courseName
12 } // end GradeBook constructor
13
14 // function to set the course name
15 void GradeBook::setCourseName( string name )
16 {
17    courseName = name; // store the course name in the object
18 } // end function setCourseName
19
20 // function to get the course name
21 string GradeBook::getCourseName()
22 {
23    return courseName; // return object's courseName
24 } // end function getCourseName
25
26 // display a welcome message to the GradeBook user
27 void GradeBook::displayMessage()
28 {
29    // call getCourseName to get the courseName
30    cout << "Welcome to the grade book for\n" << getCourseName()
31       << "!" << endl;
32 } // end function displayMessage
```

Interface - Compilation and Linkage



Interface - Deconstructor

```
1  // Fig. 17.11: CreateAndDestroy.h
2  // CreateAndDestroy class definition.
3  // Member functions defined in CreateAndDestroy.cpp.
4  #include <string>
5  using namespace std;
6
7  #ifndef CREATE_H
8  #define CREATE_H
9
10 class CreateAndDestroy
11 {
12 public:
13    CreateAndDestroy( int, string ); // constructor
14    ~CreateAndDestroy(); // destructor
15 private:
16    int objectID; // ID number for object
17    string message; // message describing object
18 }; // end class CreateAndDestroy
19
20 #endif
```

```
1  // Fig. 17.12: CreateAndDestroy.cpp
2  // CreateAndDestroy class member-function definitions.
3  #include <iostream>
4  #include "CreateAndDestroy.h"// include CreateAndDestroy class definition
5  using namespace std;
6
7  // constructor
8  CreateAndDestroy::CreateAndDestroy( int ID, string messageString )
9  {
10    objectID = ID; // set object's ID number
11    message = messageString; // set object's descriptive message
12
13    cout << "Object " << objectID << "   constructor runs   "
14       << message << endl;
15 } // end CreateAndDestroy constructor
16
17 // destructor
18 CreateAndDestroy::~CreateAndDestroy()
19 {
20    // output newline for certain objects; helps readability
21    cout << ( objectID == 1 || objectID == 6 ? "\n" : "" );
22
23    cout << "Object " << objectID << "   destructor runs    "
24       << message << endl;
25 } // end ~CreateAndDestroy destructor
```

Const Objects and Functions

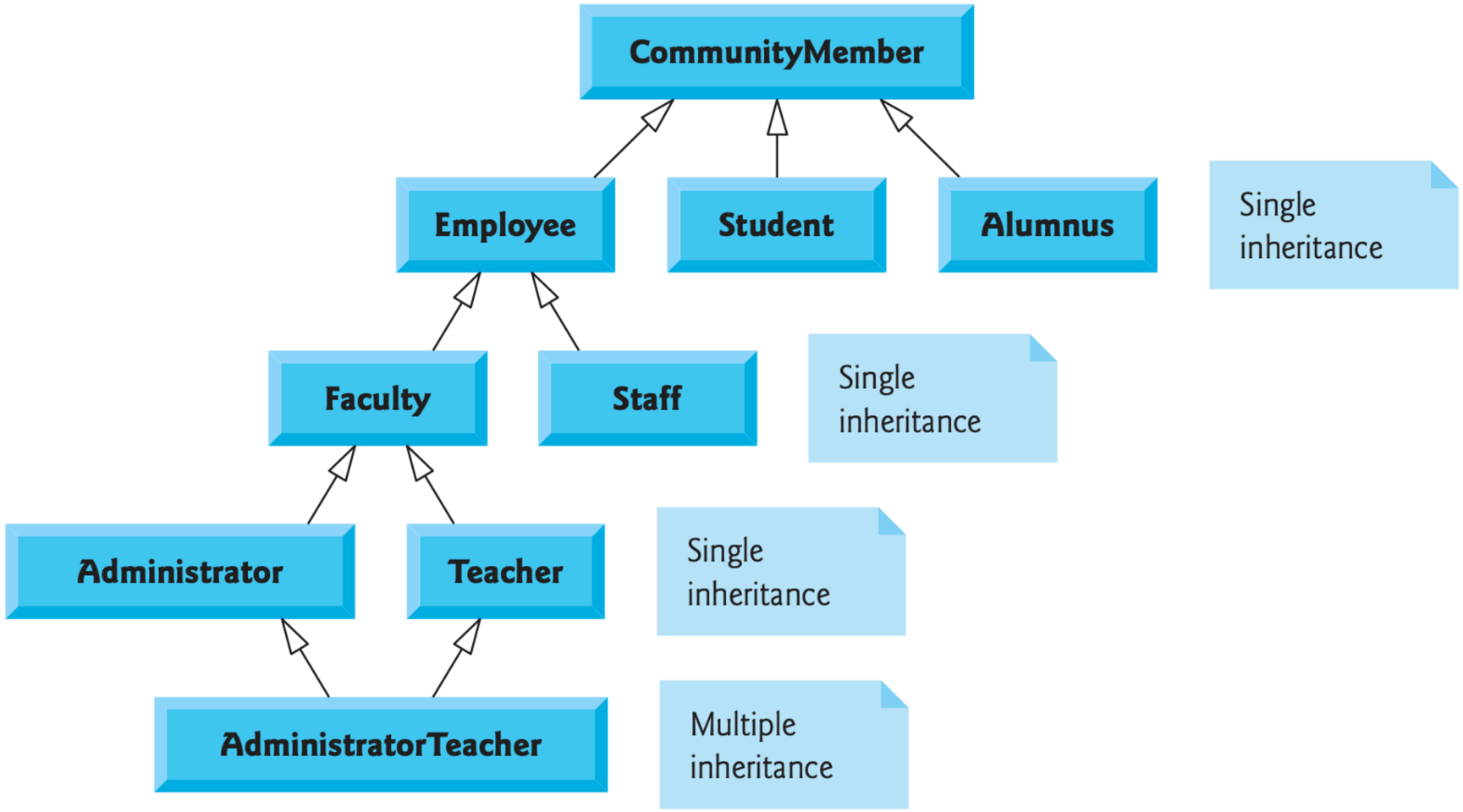
```
const Time noon( 12, 0, 0 );
```

```
C:\examples\ch18\fig18_01_03\fig18_03.cpp(13) : error C2662:
   'Time::setHour' : cannot convert 'this' pointer from 'const Time' to 'Time &'
        Conversion loses qualifiers
C:\examples\ch18\fig18_01_03\fig18_03.cpp(20) : error C2662:
   'Time::printStandard' : cannot convert 'this' pointer from 'const Time' to
   'Time &'
        Conversion loses qualifiers
```

```
 7  class Time
 8  {
 9  public:
10     Time( int = 0, int = 0, int = 0 ); // default constructor
11
12     // set functions
13     void setTime( int, int, int ); // set time
14     void setHour( int ); // set hour
15     void setMinute( int ); // set minute
16     void setSecond( int ); // set second
17
18     // get functions (normally declared const)
19     int getHour() const; // return hour
20     int getMinute() const; // return minute
21     int getSecond() const; // return second
22
23     // print functions (normally declared const)
24     void printUniversal() const; // print universal time
25     void printStandard(); // print standard time (should be const)
26 private:
27     int hour; // 0 - 23 (24-hour clock format)
28     int minute; // 0 - 59
29     int second; // 0 - 59
30 }; // end class Time
31
32 #endif
```

```
 1  // Fig. 18.3: fig18_03.cpp
 2  // Attempting to access a const object with non-const member functions.
 3  #include "Time.h" // include Time class definition
 4
 5  int main()
 6  {
 7     Time wakeUp( 6, 45, 0 ); // non-constant object
 8     const Time noon( 12, 0, 0 ); // constant object
 9
10                                // OBJECT      MEMBER FUNCTION
11     wakeUp.setHour( 18 );      // non-const   non-const
12
13     noon.setHour( 12 );        // const       non-const
14
15     wakeUp.getHour();          // non-const   const
16
17     noon.getMinute();          // const       const
18     noon.printUniversal();     // const       const
19
20     noon.printStandard();      // const       non-const
21  } // end main
```

Inheritance (Single vs. Multiple)



Inheritance

```
1  // Fig. 20.4: CommissionEmployee.h
2  // CommissionEmployee class header.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // C++ standard string class
7  using namespace std;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14
15    void setFirstName( const string & ); // set first name
16    string getFirstName() const; // return first name
17
18    void setLastName( const string & ); // set last name
19    string getLastName() const; // return last name
20
21    void setSocialSecurityNumber( const string & ); // set SSN
22    string getSocialSecurityNumber() const; // return SSN
23
24    void setGrossSales( double ); // set gross sales amount
25    double getGrossSales() const; // return gross sales amount
26
27    void setCommissionRate( double ); // set commission rate (percentage)
28    double getCommissionRate() const; // return commission rate
29
30    double earnings() const; // calculate earnings
31    void print() const; // print CommissionEmployee object
32 private:
33    string firstName;
34    string lastName;
35    string socialSecurityNumber;
36    double grossSales; // gross weekly sales
37    double commissionRate; // commission percentage
38 }; // end class CommissionEmployee
39
40 #endif
```

Inheritance

```
10 class BasePlusCommissionEmployee
11 {
12 public:
13    BasePlusCommissionEmployee( const string &, const string &,
14       const string &, double = 0.0, double = 0.0, double = 0.0 );
15
16    void setFirstName( const string & ); // set first name
17    string getFirstName() const; // return first name
18
19    void setLastName( const string & ); // set last name
20    string getLastName() const; // return last name
21
22    void setSocialSecurityNumber( const string & ); // set SSN
23    string getSocialSecurityNumber() const; // return SSN
24
25    void setGrossSales( double ); // set gross sales amount
26    double getGrossSales() const; // return gross sales amount
27
28    void setCommissionRate( double ); // set commission rate
29    double getCommissionRate() const; // return commission rate
30
31    void setBaseSalary( double ); // set base salary
32    double getBaseSalary() const; // return base salary
33
34    double earnings() const; // calculate earnings
35    void print() const; // print BasePlusCommissionEmployee object
36 private:
37    string firstName;
38    string lastName;
39    string socialSecurityNumber;
40    double grossSales; // gross weekly sales
41    double commissionRate; // commission percentage
42    double baseSalary; // base salary
43 }; // end class BasePlusCommissionEmployee
44
45 #endif
```

```
1  // Fig. 20.10: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class derived from
3  // class CommissionEmployee.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  #include "CommissionEmployee.h" // CommissionEmployee class declaration
9  using namespace std;
10
11 class BasePlusCommissionEmployee : public CommissionEmployee
12 {
13 public:
14    BasePlusCommissionEmployee( const string &, const string &,
15       const string &, double = 0.0, double = 0.0, double = 0.0 );
16
17    void setBaseSalary( double ); // set base salary
18    double getBaseSalary() const; // return base salary
19
20    double earnings() const; // calculate earnings
21    void print() const; // print BasePlusCommissionEmployee object
22 private:
23    double baseSalary; // base salary
24 }; // end class BasePlusCommissionEmployee
25
26 #endif
```

Inheritance
Protected Data

```
1  // Fig. 20.12: CommissionEmployee.h
2  // CommissionEmployee class definition with protected data.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // C++ standard string class
7  using namespace std;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14
15    void setFirstName( const string & ); // set first name
16    string getFirstName() const; // return first name
17
18    void setLastName( const string & ); // set last name
19    string getLastName() const; // return last name
20
21    void setSocialSecurityNumber( const string & ); // set SSN
22    string getSocialSecurityNumber() const; // return SSN
23
24    void setGrossSales( double ); // set gross sales amount
25    double getGrossSales() const; // return gross sales amount
26
27    void setCommissionRate( double ); // set commission rate
28    double getCommissionRate() const; // return commission rate
29
30    double earnings() const; // calculate earnings
31    void print() const; // print CommissionEmployee object
32 protected:
33    string firstName;
34    string lastName;
35    string socialSecurityNumber;
36    double grossSales; // gross weekly sales
37    double commissionRate; // commission percentage
38 }; // end class CommissionEmployee
39
40 #endif
```

Polymorphism

- Assignment of derived-class address to base-class pointer
- Assignment of base-class address to derived-class pointer
- Invoking derived-class functions with base-class pointer
- Invoking base-class functions with derived-class pointer

Polymorphism (1)

```
10 int main()
11 {
12    // create base-class object
13    CommissionEmployee commissionEmployee(
14       "Sue", "Jones", "222-22-2222", 10000, .06 );
15
16    // create base-class pointer
17    CommissionEmployee *commissionEmployeePtr = 0;
18
19    // create derived-class object
20    BasePlusCommissionEmployee basePlusCommissionEmployee(
21       "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
22
23    // create derived-class pointer
24    BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;
25
26    // set floating-point output formatting
27    cout << fixed << setprecision( 2 );
28
29    // output objects commissionEmployee and basePlusCommissionEmployee
30    cout << "Print base-class and derived-class objects:\n\n";
31    commissionEmployee.print(); // invokes base-class print
32    cout << "\n\n";
33    basePlusCommissionEmployee.print(); // invokes derived-class print
35    // aim base-class pointer at base-class object and print
36    commissionEmployeePtr = &commissionEmployee; // perfectly natural
37    cout << "\n\n\nCalling print with base-class pointer to "
38       << "\nbase-class object invokes base-class print function:\n\n";
39    commissionEmployeePtr->print(); // invokes base-class print
40
41    // aim derived-class pointer at derived-class object and print
42    basePlusCommissionEmployeePtr = &basePlusCommissionEmployee; // natural
43    cout << "\n\n\nCalling print with derived-class pointer to "
44       << "\nderived-class object invokes derived-class "
45       << "print function:\n\n";
46    basePlusCommissionEmployeePtr->print(); // invokes derived-class print
47
48    // aim base-class pointer at derived-class object and print
49    commissionEmployeePtr = &basePlusCommissionEmployee;
50    cout << "\n\n\nCalling print with base-class pointer to "
51       << "derived-class object\ninvokes base-class print "
52       << "function on that derived-class object:\n\n";
53    commissionEmployeePtr->print(); // invokes base-class print
54    cout << endl;
55 } // end main
```

Polymorphism (1)

```
Print base-class and derived-class objects:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00


Calling print with base-class pointer to
base-class object invokes base-class print function:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06


Calling print with derived-class pointer to
derived-class object invokes derived-class print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00


Calling print with base-class pointer to derived-class object
invokes base-class print function on that derived-class object:

commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
```

Polymorphism (2)

```
C:\cpphtp8_examples\ch13\Fig13_02\fig13_02.cpp(14) : error C2440: '=' :
   cannot convert from 'CommissionEmployee *' to 'BasePlusCommissionEmployee *'
        Cast from base to derived requires dynamic_cast or static_cast
```

```
1  // Fig. 13.2: fig13_02.cpp
2  // Aiming a derived-class pointer at a base-class object.
3  #include "CommissionEmployee.h"
4  #include "BasePlusCommissionEmployee.h"
5
6  int main()
7  {
8     CommissionEmployee commissionEmployee(
9        "Sue", "Jones", "222-22-2222", 10000, .06 );
10    BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;
11
12    // aim derived-class pointer at base-class object
13    // Error: a CommissionEmployee is not a BasePlusCommissionEmployee
14    basePlusCommissionEmployeePtr = &commissionEmployee;
15 } // end main
```

Polymorphism (3)

```
fig13_03.cpp:26: error: 'getBaseSalary' undeclared (first use this function)
fig13_03.cpp:27: error: 'setBaseSalary' undeclared (first use this function)
```

```
 7  int main()
 8  {
 9     CommissionEmployee *commissionEmployeePtr = 0; // base class
10     BasePlusCommissionEmployee basePlusCommissionEmployee(
11        "Bob", "Lewis", "333-33-3333", 5000, .04, 300 ); // derived class
12
13     // aim base-class pointer at derived-class object
14     commissionEmployeePtr = &basePlusCommissionEmployee;
15
16     // invoke base-class member functions on derived-class
17     // object through base-class pointer (allowed)
18     string firstName = commissionEmployeePtr->getFirstName();
19     string lastName = commissionEmployeePtr->getLastName();
20     string ssn = commissionEmployeePtr->getSocialSecurityNumber();
21     double grossSales = commissionEmployeePtr->getGrossSales();
22     double commissionRate = commissionEmployeePtr->getCommissionRate();
23
24     // attempt to invoke derived-class-only member functions
25     // on derived-class object through base-class pointer (disallowed)
26     double baseSalary = commissionEmployeePtr->getBaseSalary();
27     commissionEmployeePtr->setBaseSalary( 500 );
28  } // end main
```

Exception Handling

```
1  // Fig. 16.1: DivideByZeroException.h
2  // Class DivideByZeroException definition.
3  #include <stdexcept> // stdexcept header contains runtime_error
4  using namespace std;
5
6  // DivideByZeroException objects should be thrown by functions
7  // upon detecting division-by-zero exceptions
8  class DivideByZeroException : public runtime_error
9  {
10 public:
11    // constructor specifies default error message
12    DivideByZeroException()
          runtime_error( "attempted to divide by zero" )
14 }; // end class DivideByZeroException
```

```
1  // Fig. 16.2: Fig16_02.cpp
2  // A simple exception-handling example that checks for
3  // divide-by-zero exceptions.
4  #include <iostream>
5  #include "DivideByZeroException.h" // DivideByZeroException class
6  using namespace std;
7
8  // perform division and throw DivideByZeroException object if
9  // divide-by-zero exception occurs
10 double quotient( int numerator, int denominator )
11 {
12    // throw DivideByZeroException if trying to divide by zero
13    if ( denominator == 0 )
14       throw DivideByZeroException(); // terminate function
15
16    // return division result
17    return static_cast< double >( numerator ) / denominator;
18 } // end function quotient
19
20 int main()
21 {
22    int number1; // user-specified numerator
23    int number2; // user-specified denominator
24    double result; // result of division
25
26    cout << "Enter two integers (end-of-file to end): ";
27
28    // enable user to enter two integers to divide
29    while ( cin >> number1 >> number2 )
30    {
31       // try block contains code that might throw exception
32       // and code that will not execute if an exception occurs
33       try
34       {
35          result = quotient( number1, number2 );
36          cout << "The quotient is: " << result << endl;
37       } // end try
38       catch ( DivideByZeroException &divideByZeroException )
39       {
40          cout << "Exception occurred: "
41             << divideByZeroException.what() << endl;
42       } // end catch
43
44       cout << "\nEnter two integers (end-of-file to end): ";
45    } // end while
46
47    cout << endl;
48 } // end main
```

Exception Handling

```
7  // throw, catch and rethrow exception
8  void throwException()
9  {
10    // throw exception and catch it immediately
11    try
12    {
13       cout << "  Function throwException throws an exception\n";
14       throw exception(); // generate exception
15    } // end try
16    catch ( exception & ) // handle exception
17    {
18       cout << "  Exception handled in function throwException"
19          << "\n  Function throwException rethrows exception";
20       throw; // rethrow exception for further processing
21    } // end catch
22
23    cout << "This also should not print\n";
24 } // end function throwException
25
26 int main()
27 {
28    // throw exception
29    try
30    {
31       cout << "\nmain invokes function throwException\n";
32       throwException();
33       cout << "This should not print\n";
34    } // end try
35    catch ( exception & ) // handle exception
36    {
37       cout << "\n\nException handled in main\n";
38    } // end catch
39
40    cout << "Program control continues after catch in main\n";
41 } // end main
```

Exception Handling